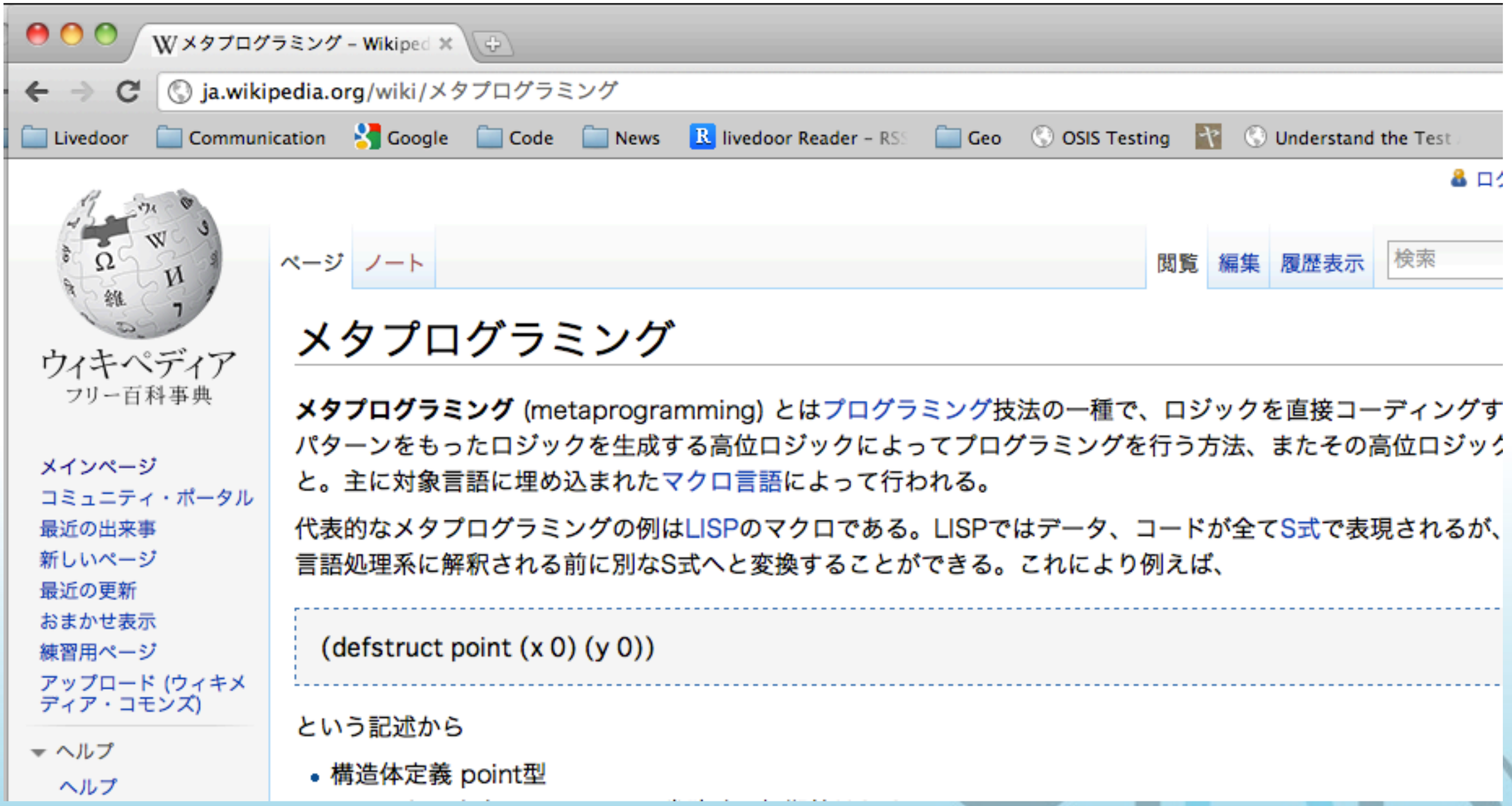
メタプログラミング
Perl編

Japan Perl Association
株式会社ライブドア
牧　大輔

- @lestrrat
- 12 yrs of writing perl
- Internal Tools Guy
- JP→BR→JP→PT→US→JP

特にこれが専門という
わけではないので

調べてみた


メタプログラミング - Wikiped
ja.wikipedia.org/wiki/メタプログラミング
Livedoor
Communication
Google
Code
News
livedoor Reader - RSS
Geo
OSIS Testing
Understand the Test
ページ
ノート
閲覧
編集
履歴表示
検索
ウィキペディア
フリー百科事典
メインページ
コミュニティ・ポータル
最近の出来事
新しいページ
最近の更新
おまかせ表示
練習用ページ
アップロード (ウィキメディア・コモンズ)
ヘルプ
ヘルプ
メタプログラミング
メタプログラミング (metaprogramming) とはプログラミング技法の一種で、ロジックを直接コーディングす
パターンをもったロジックを生成する高位ロジックによってプログラミングを行う方法、またその高位ロジック
と。主に対象言語に埋め込まれたマクロ言語によって行われる。
代表的なメタプログラミングの例はLISPのマクロである。LISPではデータ、コードが全てS式で表現されるが、
言語処理系に解釈される前に別なS式へと変換することができる。これにより例えば、
(defstruct point (x 0) (y 0))
という記述から
構造体定義 point型

要約：
プログラムを生成する
プログラムね？

なんとなく探った結果

evalとかも入るけど、DSL作るとかメタオブジェクトプログラミングとかがうけるらしい

まず簡単なeval系

```
my $str = “sub {\n”;
foreach my $word ( qw(foo bar baz) ) {
  $str .= “    return 1 if \$_[0] =~ /$word/;\n”;
}
$str .= “    return;\n}\n”;

my $code = eval $str;
$code->(“foo”);  # TRUE
$code->(“bar”);  # TRUE
$code->(“fuga”); # FALSE
```

動的に任意の文字列にだけマッチする関数を作る、とか

まず簡単なeval系

```
my $str = <<EOM;
sub {
    return 1 if \$_[0] =~ /foo/;
    return 1 if \$_[0] =~ /bar/;
    return 1 if \$_[0] =~ /baz/;
    return;
}
EOM

my $code = eval $str;
$code->(“foo”);   # TRUE
$code->(“bar”);   # TRUE
$code->(“fuga”); # FALSE
```

動的に任意の文字列にだけマッチする関数を作る、とか

普通のクラス

```
package MyClass;
use strict;
use base qw(Class::Accessor::Fast);

sub foo { ... }
sub bar { ... }
```

皆さんの大好きなPerlクラスのテンプレート

めたぷろぐらみんぐ
してみたい！

eval()でよくね（ｒｙ

```
my $classname = “MyClass”;
my @superclasses = qw(Class::Accessor::Fast);

eval <<EOM;
   package $classname;
   use base qw(@superclasses);
EOM
```

考え方は一番ストレート

メソッド定義あたりでエスケープの嵐で破綻する

グロブ・グローバル変数制御

```
@MyClass::ISA = qw( Class::Accessor::Fast );
# 継承

*MyClass::foo = sub { ... }; # メソッド宣言

sub MyClass::bar { ... } # また違うメソッド宣言
```

名前空間に関数、変数をくっつける事は自由
これだけでほぼ全ての事ができる

え、そんなMOP嫌だ？

でもPerlはその辺りが可愛いのに・・・

わがままさん☆

Moose

```
package MyClass;
use Moose;

# 継承
extends 'FooBar';
# ロール適用
with 'WithHoge';

# アトリビュート宣言。初期化用のコードやアクセサーも作成
has foo => ( is => 'ro', isa => 'Int' );
```

クラスを定義するためのDSLの塊
言語にないものはライブラリで作る！それがPerl!

MooseでMOP

```
my $meta = Moose::Meta::Class->create(
    “MyClass”,
    superclasses => [ “FooBar” ]
);
$meta->add_method( “hoge” => sub { ... } );
$meta->add_attribute( “foo” => (
    is => ‘ro’
    isa => ‘Int’,
) );
my $object = $meta->new_object();
```

クラスを定義するためのDSLの塊

無名クラス
(テストとかに結構便利)

```
# MyClassを継承する無名クラスを作成
my $meta = Moose::Meta::Class->create_anon_class(
    superclasses => [ “MyClass” ]
);

# 親クラスのhogeメソッドをラップしたメソッドを定義
$meta->add_around_method_modifier( hoge => sub { ... } );
```

特異クラス等の実装に！

あ、DSLのほうが
いいんでしたっけ？

PerlでDSL（簡単編）

```
sub declare_as ($@) { # <--- “prototype”
    ...
}

sub bar ($) { ... }
```

他のどこでもほぼ使うアテの内Perl
の”prototype”を使うとちょっと楽しい

PerlでDSL（基本）

```
declare_as ‘hoge’, ( opt => “val”, opt2 => “val” );

bar ‘fuga’, ‘fuga’; # 一応これでもOK・・・
                    # ただし、($)としか指定していないので、
                    # bar(‘fuga’), ‘fuga’ とパースされる
```

$や@で引数のコンテキスト・構文が
曖昧な時の結びつきを指定できる

PerlでDSL（飛翔編）

```
# こういうのつくりたい！

map { ... コード ... } @list;
```

PerlでDSL（飛翔編）

```
sub mymap { ... }

# syntax error "} @list"
mymap { ... コード ... } @list;
```

PerlでDSL（飛翔編）

```
sub mymap { ... }

# syntax error “} @list”
mymap { ... コード ... }, @list;
```

ちなみにカンマをいれると、{}はコードブロックで
はなく、無名ハッシュとして認識される

PerlでDSL（飛翔編）

```
# prototype '&'
sub mymap (&@) { ... }

# OK!
mymap { ... コード ... } @list;
```

PerlでDSL（飛翔編）

```
sub mymap (&@) {
    my ($code, @list) = @_;
    my @result;
    foreach my $arg (@list) {
        push @result, $code->($arg);
    }
    return @result;
}
```

&を指定すると、subキーワードを
使わずにコードブロックを指定することが可能

PerlでDSL（飛翔編）

```
use Teng::Schema::Declare;
my $schema = schema { # Start block
      table { # Another block
            name ‘tablename’;
            pk ‘primary_key’;
            columns qw(hoge fuga moga);
      }
} “MySchema”;
```

なんかかっこいい！

無名コードブロックでコードをネストしていける

注意事項

&指定は第一引数に指定した時のみ、sub宣言を省略できる。また@を第一引数として指定してしまうとそれが残りの引数を全て食ってしまうので+という表現を使わないとpushのような形の引数宣言はできないので、これを使う際には@_の中身に充分注意すること

要約：perldoc perlsubを良く読め

PerlでDSL（変態編）

```
use 5.10;
use Moose;
use MooseX::Method::Signatures;

# !!! method宣言？？ 引数の型指定？

method hello ( Int $howmany ) {
    $self->log_info( join “ “, (“hello”) x $howmany );
}

method log_info ( Str $message ) {
    say $message;
}
```

Perlにmethodなんてないんだけど！

PerlでDSL（変態編）

- Devel::Declare
  - Perlのパーサーにフック
  - PerlだけでPerlの構文を変えられる
  - 究極のDSL作成ツール！

PerlでDSL（変態編）

- Devel::Declare
  - 重い
  - きもい
  - 某氏：「Devel::Declare のソースよめない人はつかわない」
- 詳細は自分でみてね！
  - http://api.metacpan.org/source/FLORA/Devel-Declare-0.006005/t/method.t

結論１：Perlは言語がそもそもメタ

Perlは言語の構文を変えてまで DSLやらメタな事ができる

結論2：普通に書け

メタプログラミングやってて
Perlの事を悪く言っちゃ駄目

メタプログラミングやってて
Perlの事を悪く言っちゃ駄目
YOU ASKED FOR IT.

細かい所はPerlハッカー達に
YAPCで聞いてね！

YAPC::ASIA TOKYO 2011

Welcome!

今年もやってきましたYAPC::Asia Tokyo 2011！今年は 10/13（仮：前夜祭）, 14, 15に執り行われます。会場はおなじみ東京工業大学 大岡山キャンパスです。

スポンサーも絶賛募集中ですので、ご興味のあるかたは info-at-perlassociation.org までご連絡ください。

News

Sponsors

主催

JPA

スポンサー

ただいまスポンサー募集中です！info-at-perlassociation.orgまでご連絡を！

IRC/Tags

公式タグ、ハッシュタグは ***yapcasia*** を使用してください。ブックマーク、ブログ、写真等のタグ、そしてTwitterでご使用いただけると情報共有に便利です。年度を特定したい場合は ***yapcasia2011*** でも結構ですが、***yapcasia*** と併用してくださると助かります。

また、IRCは freenode上の ***#yapcasia-en*** (英語)と ***#yapcasia-ja*** です。

Socials

YAPC::ASIA TOKYO 2011

Welcome Access/Date Guest Speakers News English

Sponsors

主催

2011 10.13 ~ 10.15

http://yapcasia.org

please book your flights!

Welcome!

今年もやってきましたYAPC::Asia Tokyo 2011！今年は 10/13（仮：前夜祭）, 14, 15に執り行われます。会場はおなじみ東京工業大学 大岡山キャンパスです。

スポンサーも絶賛募集中ですので、ご興味のあるかたは info-at-perlassociation.org までご連絡ください。

公式タグ、ハッシュタグは *yapcasia* を使用してください。ブックマーク、ブログ、写真等のタグ、そしてTwitterでご使用いただけると情報共有に便利です。年度を特定したい場合は *yapcasia2011* でも結構ですが、*yapcasia*と併用してくださると助かります。

また、IRCは freenode上の *#yapcasia-en* (英語)と *#yapcasia-ja* です。

News

Sociales

ありがとうございました